HACKING WITH SWIFT

SWIFTDATA BY EXAMPLE

COMPLETE REFERENCE GUIDE

Get hands-on solutions for common problems

Paul Hudson

Chapter 1
Introduction

A brief explanation of the basics of SwiftData

What is SwiftData?

SwiftData is a fast, powerful, and easy-to-use way to store data in apps built for iOS, macOS, tvOS, watchOS, and even visionOS. It lets us create custom objects, define how they link together, retrieve them with filtering and sorting, and even synchronize them to iCloud – and much more too.

Not only does SwiftData take full advantage of the latest Swift language features, but it's also built with SwiftUI in mind: if you're building apps with SwiftUI, you'll find SwiftData slots in almost invisibly.

Behind the scenes, SwiftData is powered by a much bigger and more mature framework called Core Data. That brings all sorts of benefits, not least 20 years of development and maturity. But SwiftData is more than just a simple overlay: Apple really went to town in isolating and resolving the key pain points developers were reporting with the older framework, meaning that SwiftData is a significant improvement for anyone who has used Core Data in the past.

One downside is that SwiftData supports only iOS 17 or later, along with other coordinated releases – that's macOS Sonoma, tvOS 17, watchOS 10, and visionOS 1.0.

SwiftData makes a great choice for any kind of on-device storage, including:

1. Permanent storage of user data, such as their to do lists or cooking recipes.
2. Temporary storage of user data, where SwiftData is used as a cache for data fetched from a server.
3. Document-based apps, e.g. text or video editors.
4. Complex user settings or history data.

It's *less* of a great choice when:

1. You need to support many users using iOS 16 and earlier. Although SwiftData and Core Data can live side by side by in the same app, it's extra work.
2. Your data is stored only in CloudKit or another equivalent service, and you need to be using live data at all times.

3. You need the full range of capabilities offered by Core Data. Many features from Core Data have yet to surface in SwiftData, so if you have more advanced use-cases you should probably stick with Core Data for now.

SwiftData vs Core Data

Although SwiftData builds on top of Core Data, not all the functionality has been exposed for us to use. This means quite a few major Core Data features are not yet supported for developers working exclusively in SwiftData, including:

1. We don't have an equivalent of **NSCompoundPredicate**, for creating complex, multi-step predicates.
2. We don't have an equivalent of **NSFetchedResultsController**, for executing then monitoring queries for changes.
3. There is no support for derived attributes, so things like an automatic **lastUpdated** property aren't possible.
4. There are no sectioned fetched requests.
5. SwiftData does not support abstract classes or child contexts.
6. Or pinning to a specific query generation.

Everyone will have different priorities for those features, but for me missing the first two is *hard*. Hopefully we'll see SwiftData continue to improve rapidly over time, until we eventually reach full parity with Core Data.

You should also keep in mind that SwiftData is extremely new, whereas Core Data has been in constant development for about 20 years. This means you're likely to hit edge cases and surprises on a fairly frequent basis, and I think it's fair to say that SwiftData has more sharp edges than a porcupine's dance party. Do things exactly right and you'll be happy, but a lot of the time you'll find yourself staring at a crash wondering which small but apparently critical mistake you made.

Don't despair – it will get better! In the meantime, please watch out for the many things marked **Important** or **Tip** in this guide; I've gone through the pain for you, so please save yourself the hassle and learn from that pain rather than repeating it.

Should you learn SwiftData, Core Data, or both?

Because SwiftData is an overlay over Core Data, deciding to learn either SwiftData or Core Data is a fairly easy one: if you are able to live without the missing features such as **NSFetchedResultsController**, and you are able to target iOS 17 and later, learning SwiftData is significantly faster and easier – you'll be building apps in a tenth the time it would take for you to learn Core Data.

However, the nice thing about its overlay status is that it's surprisingly easy to move from SwiftData to Core Data without risking any user information. This means if you're six months into a project and realize you just can't live without one particular feature, you can move your code across to the older framework without too much hassle.

One slightly less obvious downside to SwiftData is that its newness means it has significantly less documentation available, both from Apple but also the community. This means if you have a problem you'll have a harder time finding solutions, simply because there are fewer people asking and answering on sites such as Stack Overflow. Hopefully this book goes a long way to filling the gaps!

In the meantime, one thing you can be clear on is Apple's direction of travel: Swift, SwiftUI, and SwiftData are the best way to build apps for Apple's platforms, and will continue to expand for years to come.

One signal for this is the logos: Swift, SwiftUI, and SwiftData all have logos that incorporate the Swift bird, which I think is intentional – I think it's Apple's way of saying these three form the new Cocoa. In the past, Cocoa meant using Foundation, AppKit, and Core Data, but now we have the Swift standard library, SwiftUI, and SwiftData, all of which combine to make app development as enjoyable, efficient, and safe as it can be.

Frequently asked questions about SwiftData

Ever since SwiftData was announced folks have been asking a whole bunch of questions to me, to Apple engineers, and more. I've answered as many of those as I can in this book, but the questions below are asked particularly commonly:

Is it hard to start using SwiftData?

Core Data had quite a difficulty ramp for getting set up properly, but SwiftData takes such a small amount of code you'll almost think you've missed something! Even when you start venturing into more advanced features, SwiftData does a great job of progressive disclosure – you can learn it bit by bit and get benefit at each step of the way, rather than having to learn many things at once just to get moving.

Can Core Data apps be migrated to SwiftData?

Yes! SwiftData uses Core Data under the hood, which means all your data structures and more will remain intact. This means if you've shipped a Core Data app already, you can move over to SwiftData either partly or entirely whenever you're ready.

Can Core Data and SwiftData exist in the same app?

Yes, you can have both stacks running at the same time. Heck, they can even be pointing to the same data, and will stay in sync. Please make sure you keep the two data models in sync, though – if you adjust your Core Data model, make sure you apply the same change to your SwiftData model too.

Do @Model classes need to be marked final?

No, but if you try subclassing them you're going to cause a lot of pain for yourself. If you want to avoid temptation, make your classes final.

Why doesn't SwiftData use structs for data models?

We all know Swift and SwiftUI developers love using structs as a simple and efficient way to represent data, but with data it's complicated: if we load an array of users, then pass one user around for editing, we need to be able to keep all the screens in our app up to date as the user makes changes.

Sure, we could try to pass around an object identifier and just that to refer to an object, but if you think about it you're kind of just recreating pointers – it's a lot of extra effort just to land up with more or less the same result.

That's not to say SwiftData must *only* use classes: you can incorporate any kind of **Codable** data into your models, including both structs and enums, and SwiftData will ensure they are saved and loaded correctly.

Is there a way to force a CloudKit sync?

No; Apple really don't want us trying to do this.

Is there a way to add Codable support to a SwiftData object?

Yes, but it takes a little thinking – we don't get automatic **Codable** support, and instead we need to implement it ourself.

Can SwiftData be used with Objective-C?

No. Not only does it use native Swift classes, but it also uses a range of advanced Swift language features that aren't available in Objective-C.

How to follow this quick start guide

This guide is called SwiftData by Example, because it focuses particularly on providing as many examples as possible, with each one solving real problems you'll face every day.

I have tried to structure this so that most entries starts with "How to…" because this is about giving you hands-on code you can use in your own projects immediately. That also means I've tried to get to the point as fast as possible and stay there, so if you're looking for a longer, slower introduction to SwiftData I'm afraid this isn't it.

Already got some experience?

If you've already grabbed the basics of SwiftData and just want code that solves your problems, by all means just jump in wherever interests you.

My code examples are specifically written for folks who are following along more or less linearly, so if you're want to make those changes you may need to do a little light editing to make it fit your code.

Just starting out?

If you're just starting out with SwiftData you should start by completing the initial sample project, then just skip around based on what interests you.

I would recommend against using Xcode's SwiftData template. It doesn't add a great deal of helpful code, and you'll just need to replace it with your own code anyway.

Migrating from Core Data to SwiftData

If you've used Core Data before, many of the classes and concepts you know and love map pretty much directly to their SwiftData equivalents, albeit with simpler names and always without the **NS** prefix.

Here's a list to get you started, with Core Data names followed by SwiftData names:

- **NSPersistentContainer**: **ModelContainer**
- **NSPersistentCloudKitContainer**: **ModelContainer** with iCloud enabled.
- **NSManagedObjectContext**: **ModelContext**
- **NSManagedObject**: the **PersistentModel** protocol and **@Model** macro
- **NSPredicate**: the **#Predicate** macro
- **NSFetchRequest**: **FetchRequest**
- **NSFetchDescriptor**: **FetchDescriptor**
- **NSCompoundPredicate**: Has no equivalent yet
- **NSSortDescriptor**: **SortDescriptor**
- **NSMigrationStage**: **MigrationStage**
- **NSEntityMigrationPolicy**: **SchemaMigrationPlan**

Most notably, Xcode's model editor is no longer needed now – it's all code.

Dedication

Ever since SwiftUI shipped, folks in the community were asking the same question: when will Core Data get its own SwiftUI-like rethink?

It might sound easy from our perspective as external developers, but the truth is that taking something as mature and powerful as Core Data and transforming it for modern Swift is a gigantic undertaking.

We wanted to keep all the power features of Core Data, such as data migration, iCloud support, faulting, and more, but we also wanted a significantly simpler API, we wanted tight integration with SwiftUI, and above all we wanted something "Swifty" – something that used the Swift language to its fullest, so it could be as expressive as possible.

Apple announced SwiftData at WWDC23, and it has proved to be a gigantic step forward for app developers. The team behind it have managed to blend the power of Core Data with the simplicity and clarity of Swift, and when you start using it you'll realize just how amazing SwiftData is – it's often almost *invisible*, which is remarkable.

Getting to this point took an astonishing amount of effort from the SwiftData team, from the Core Data team, from the Swift and SwiftUI teams, plus developer publications and so many more.

I wish I could list them all here and thank them personally, but the only ones I can be sure of are the people who have appeared in recent WWDC videos or who have mentioned their involvement publicly.

So, this book is dedicated to Daniel Duan, Nick Gillett, Debbie Goldsmith, Luvena Huo, Scott Perry, Matt Ricketson, Jeremy Schonfeld, David Stites, Ben Trumbull, Julia Vashchenko, Rishi Verma, and all the dozens of other folks who worked so hard to make SwiftData what it is today. This absolutely includes the many people who have contributed to Core Data to make it what it is today, all the way back to Bill Bumgarner, because SwiftData is truly standing on the shoulders of giants.

We may never know how many more folks from around Apple helped make SwiftData what it is, but I hope every one of them feels proud to have helped bring it to life.

Thank you.

PS: I was there at Apple Park when Apple announced SwiftData. Having spent a decade teaching folks how to use Core Data, seeing SwiftData finally becoming a reality was a huge moment because I knew how much it would impact learners. If you'd like to see just *how* huge, I captured it live, just for you…